



# Angular Advanced - Smart components & View components



Peter Kassenaar  
[info@kassenaar.com](mailto:info@kassenaar.com)

*“In Angular, all components are equal. There is no sense of different ‘types’ of components”*



# Smart components / View Components

- Design pattern
- Why ? Separation of concerns
  - **View component** is responsible for **presentation** (and *can* be used in a completely different environment with different component logic)
  - **Smart component** is responsible for **logic** → passing the [calculated] data to the view component.
- Smart components
  - AKA: *Container* components, *controller* components, *statefull* components
- View components
  - AKA: *pure* components, *dumb* components, stateless

# Characteristics

- Smart components
  - Typically contain big(ger) chunks of logic
  - Typically have a small UI part (reference just the view component)
  - Pass the data to the view component
- View components
  - Typically contain no logic whatsoever
  - Get their data via `@Input()` decorators
  - Submit events via `@Output()` decorators
  - Have large chunks of UI

# Example: ../300-smart-view-component

Home (default component)

Smart/View component

Default component- My favorite cities:


1 - Groningen	<input checked="" type="checkbox"/> Visited
2 - Hengelo	<input type="checkbox"/> Visited
3 - Den Haag	<input type="checkbox"/> Visited
4 - Enschede	<input checked="" type="checkbox"/> Visited
5 - Heerlen	<input type="checkbox"/> Visited

I Visited:

Groningen

Enschede

Current city: Groningen



# Home – Default component

- Contains all logic and UI, combined in one component.
  - /home/home.component.html
  - /home/home.component.ts
- This typically works well for smaller applications

```
<div class="row">
  <div class="col-md-6">
    <h2>Default component- My f
    <ul class="list-group">
      <li *ngFor="let city of
        class="list-group-item
        [class.visited]="city
        (click)="getCity(city
        {{ city.id}} - {{ cit
        <span class="pull-right
          <label>
        </ul>
    ..
  </div>
```

```
@Component({
  selector    : 'app-home',
  templateUrl: './home.component.html'
})
export class HomeComponent implements OnInit {
  ...
  ngOnInit() {
    this.cityService.getCities()
      .subscribe(cities => this.cities = cities);
  }

  getCity(city: City) {
    this.currentCity = city;
    this.cityPhoto   = `assets/img/${this.currentCity.name}.jpg`;
  }
  ...
}
```

# Splitting it up in view components

- `../smart-view/smart.component.html | .ts.`
- Passing `[cities]` in
- Getting (events) out.
- `[cities]` can also be an Observable

```
<div class="row">
  <div class="col-md-6">
    <h2>Smart/view component- My favorite cities:</h2>
    <!-- The <city-list>-component is now
         the view component for list of cities -->
    <city-list [cities]="cities"
              (selectCity)="getCity($event)"
              (toggleVisited)="toggleCity($event)">
    </city-list>
    ...
  </div>
</div>
```

# <city-list> View component

- Has no logic, just `@Input()`'s and `@Output()`'s

```
import {Component, EventEmitter, Input, Output} from '@angular/core';  
import {City} from '../../shared/model/city.model';
```

```
@Component({  
  selector    : 'city-list',  
  templateUrl: './city-list.component.html'  
})
```

```
export class CityListComponent {
```

```
  @Input() cities: City[];
```

```
  @Output() selectCity: EventEmitter<City> = new EventEmitter<City>();
```

```
  @Output() toggleVisited: EventEmitter<City> = new EventEmitter<City>();
```

```
}
```



# HTML of the view component

- `.../smart-view/city-list/city-list-component.html`
- Has a nested view component to toggle `visited` state
- Choice: events are directly emitted from the HTML
  - Can also be done via small functions

```
<ul class="list-group">
  <li *ngFor="let city of cities"
    class="list-group-item"
    [class.visited]="city.visited"
    (click)="selectCity.emit(city)">
    {{ city.id }} - {{ city.name }}
    <city-visited [visited]="city.visited"
      (toggle)="city.visited = $event; toggleVisited.emit(city)">
    </city-visited>
  </li>
</ul>
```



## <city-visited> View component

Again – just @Input() 's and @Output() 's.

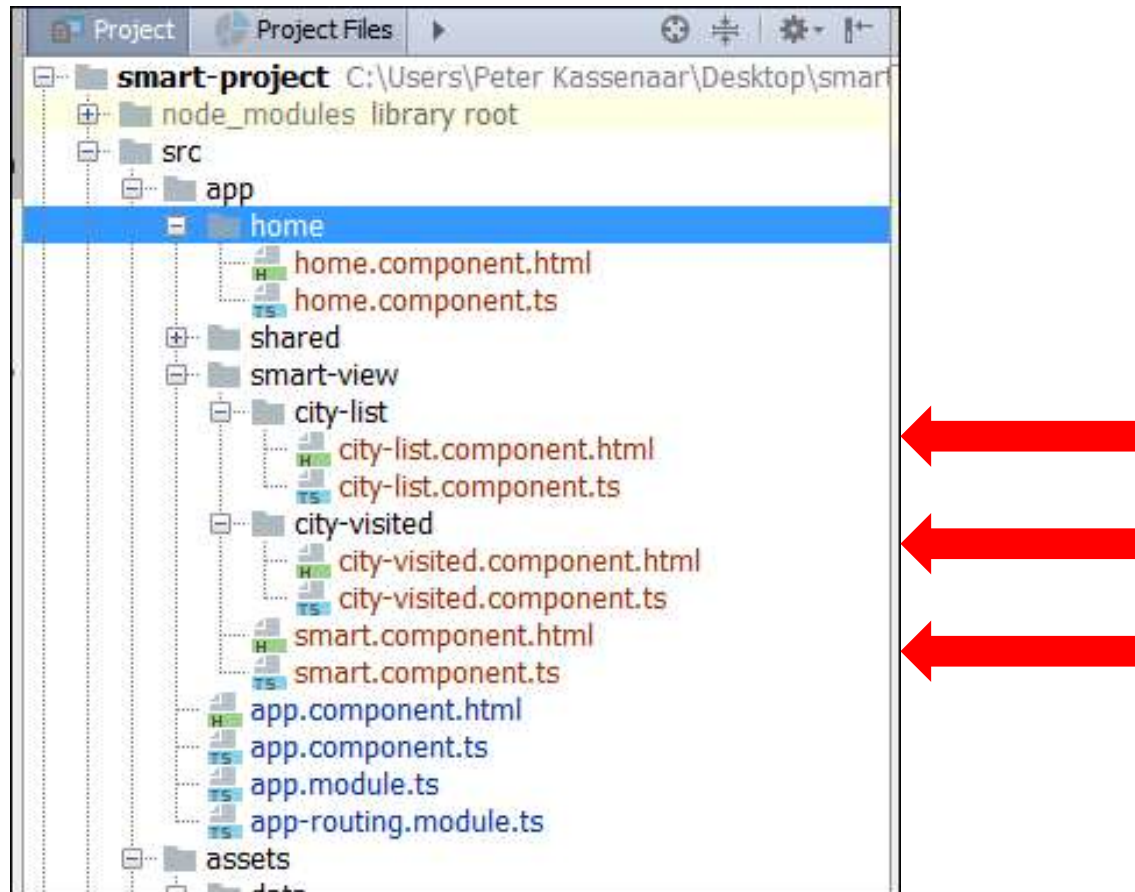
```
export class CityVisitedComponent {  
  @Input() visited: boolean;  
  @Output() toggle: EventEmitter<boolean>= new EventEmitter<boolean>();  
}
```

HTML – again: attribute binding and event binding

```
<span class="pull-right">  
  <label>  
    <input type="checkbox"  
      class="checkbox-inline"  
      [checked]="visited"  
      (change)="visited = !visited; toggle.emit(visited)"> Visited  
  </label>  
</span>
```

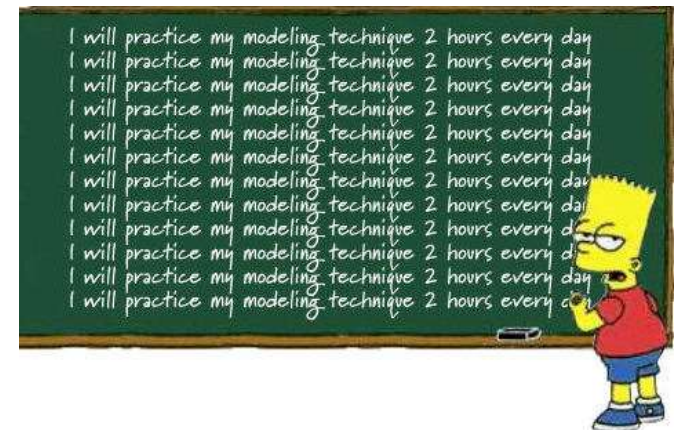
# Application structure

- When following this pattern: typically more, smaller components
- The directory tree gets crowded!

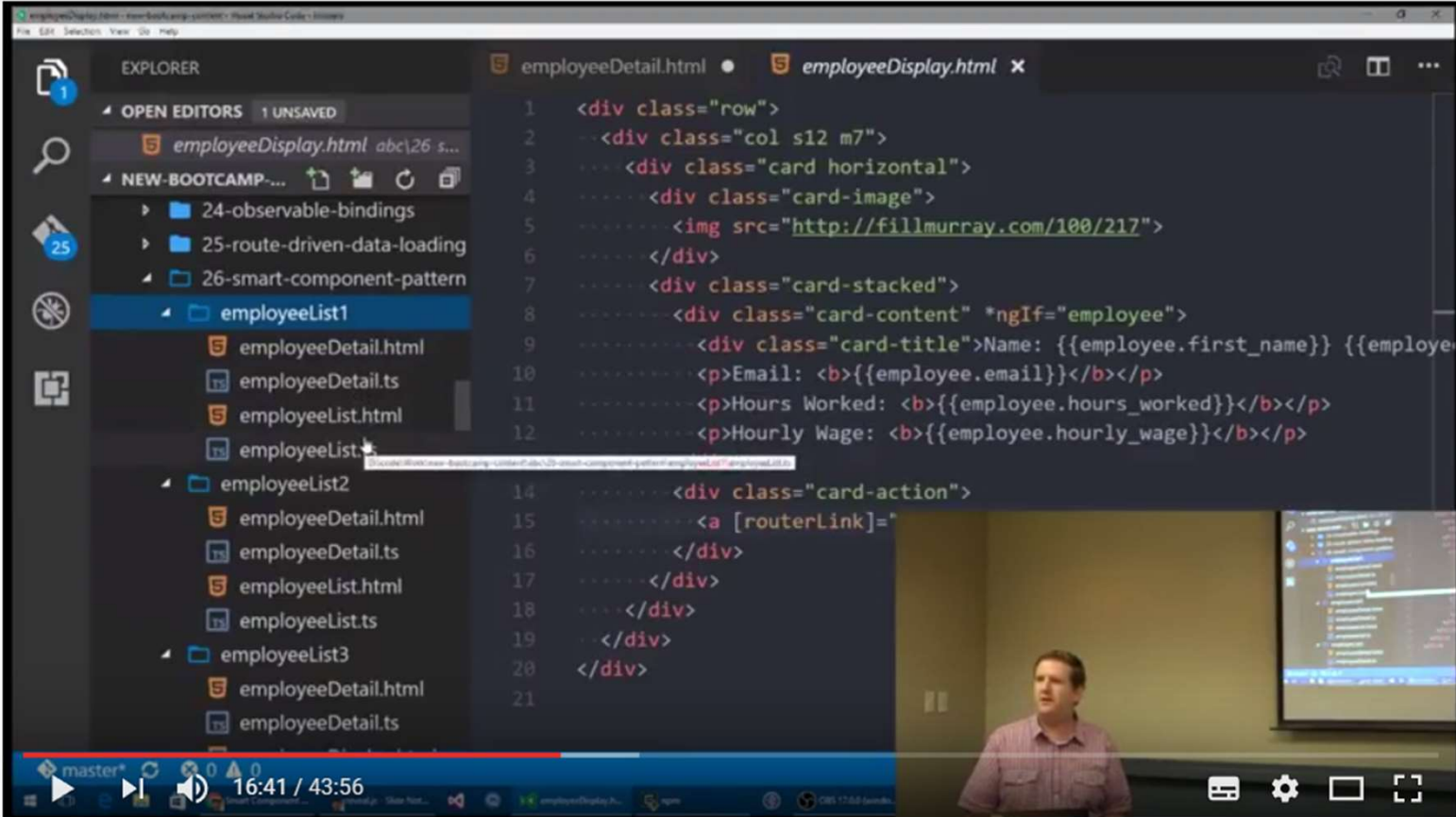


# Workshop

- Study the example `../300-smart-view-components`
- Create two view additional components:
  - One for displaying the current city
  - One for displaying the list of visited cities
- Optional: use Observables instead of plain arrays.
  - `cities$ : Observable<City[]>`
  - `[cities]="cities$ | async"`



# More info on Smart/View components



The screenshot shows a video player interface. The main content area displays a VS Code editor window. On the left, the Explorer sidebar shows a project structure with folders '24-observable-bindings', '25-route-driven-data-loading', and '26-smart-component-pattern'. Under '26-smart-component-pattern', there are subfolders 'employeeList1', 'employeeList2', and 'employeeList3'. The 'employeeList1' folder is selected, showing files 'employeeDetail.html', 'employeeDetail.ts', 'employeeList.html', and 'employeeList.ts'. The main editor area shows the 'employeeDetail.html' file with the following code:


```
1 <div class="row">
2   <div class="col s12 m7">
3     <div class="card horizontal">
4       <div class="card-image">
5         
6       </div>
7       <div class="card-stacked">
8         <div class="card-content" *ngIf="employee">
9           <div class="card-title">Name: {{employee.first_name}} {{employee
10          <p>Email: <b>{{employee.email}}</b></p>
11          <p>Hours Worked: <b>{{employee.hours_worked}}</b></p>
12          <p>Hourly Wage: <b>{{employee.hourly_wage}}</b></p>
13        </div>
14        <div class="card-action">
15          <a [routerLink]="
16        </div>
17      </div>
18    </div>
19  </div>
20 </div>
21
```

Below the code, a small video inset shows a man speaking. The video player controls at the bottom indicate the video is at 16:41 / 43:56.


Smart/View Component Patterns - St. Louis Angular Lunch - Paul Spears Jan 2017

Oasis Digital Solutions Inc.

[https://www.youtube.com/watch?v=ALm\\_JVdLT2E](https://www.youtube.com/watch?v=ALm_JVdLT2E)



**Todd Motto**  
Owner, Ultimate Angular

 Follow @toddmotto 

35K followers

≡

Blogs

👤

About

🎤

Speaking events

📺

Online courses

🔗

🎓

Workshops

🔗

🐦

🔗

📺


📺

📺


📺





© 2017

Search posts




 **ULTIMATE ANGULAR™**

**Master Angular with my online courses**

 Award winning courses trusted by thousands, there's no better place to learn

Categories:  Angular  AngularJS  RxJS  JavaScript

## Stateful and stateless components, the missing manual


 Oct 12, 2016  10 mins read  Edit post


The goals of this article are to define what stateful and stateless components are, otherwise known as smart and dumb - or container and presentational components. For the purposes of the article, we'll be using Angular 2 Components to explain the stateful/stateless concepts. Bear in mind these concepts are not at all limited to Angular, and live in other libs/frameworks such as React.

Table of contents

LATEST COURSE

Angular Fundamentals





Angular v4+ award-winning courses, learn at your own pace online with me.

<https://toddmotto.com/stateful-stateless-components>

16

 [BLOG HOME](#) [COURSES](#) [EBOOKS](#) [TUTORIALS](#) [FREE COURSE](#)

  
**Angular**  
University

  
  
  
45

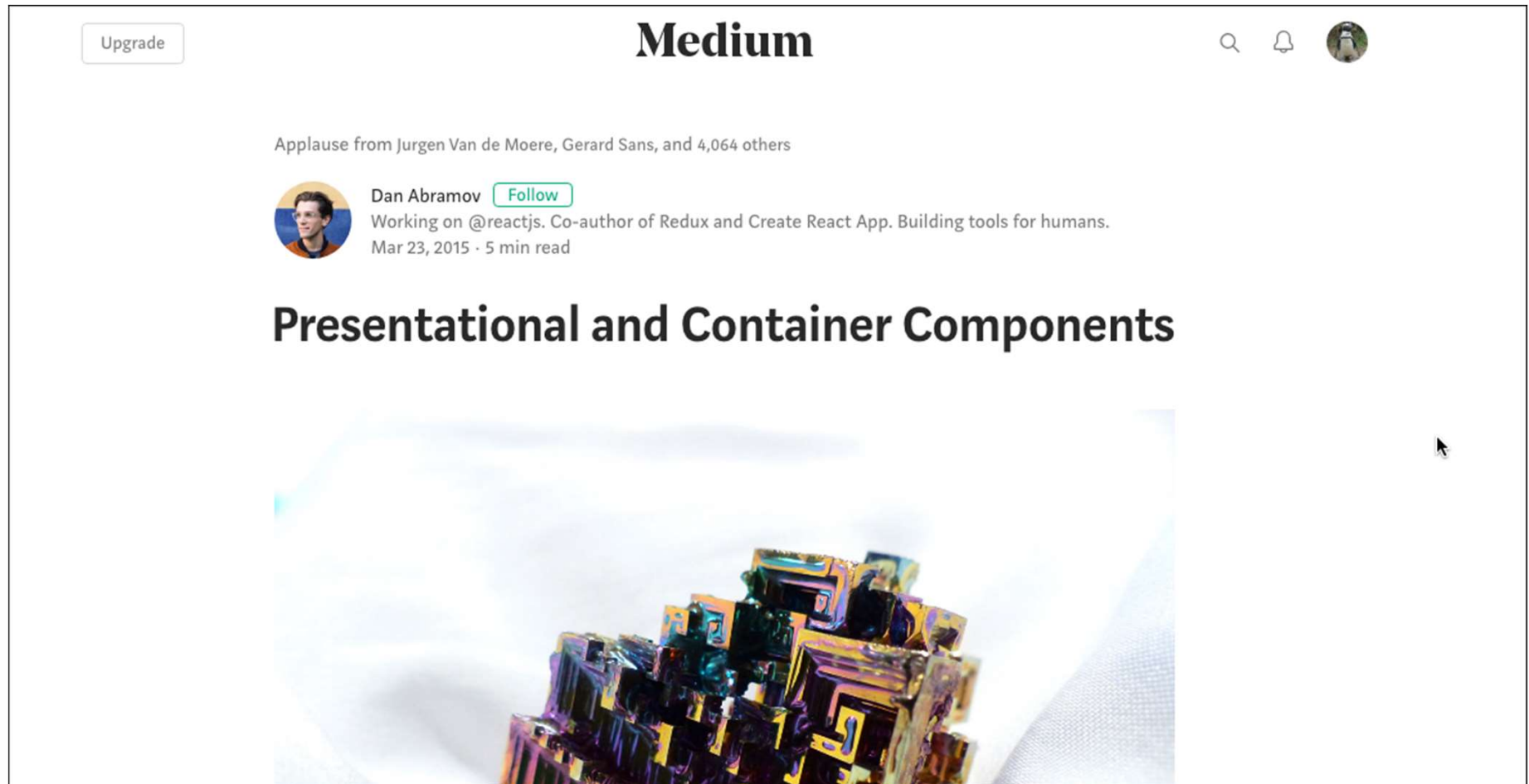
# Angular Smart Components vs Presentation Components: What's the Difference, When to Use Each and Why?

21 OCTOBER 2016

When building an Angular application, one the most frequent questions that we are faced with right at the beginning is: how do we structure our application?

<http://blog.angular-university.io/angular-2-smart-components-vs-presentation-components-whats-the-difference-when-to-use-each-and-why/>





[https://medium.com/@dan\\_abramov/smart-and-dumb-components-7ca2f9a7c7d0](https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0)





Upgrade



# How to write good, composable and pure components in Angular 2+

Most of us know what Smart and Dumb components are. We know we should use `@Input()` and `@Output()` as much as possible. But when our SPA gets big enough, it starts to remind us more and more of a typical spaghetti and it seems we cannot even help it.



Jack Tomaszewski

Follow

Jun 7, 2018 · 14 min read

**The reason is very often we know what are the good and bad patterns in the code development, but, especially in the Front-end, we often get confused with what we are ought to and not ought to do.** The patterns we should follow start to get blurry and we end up using shortcuts in our code more often than we don't.

One of such patterns is to split your components into "Smart" and "Dumb". It says that we should keep all business logic and side effects in the Smart

<https://medium.com/@jtomaszewski/how-to-write-good-composable-and-pure-components-in-angular-2-1756945c0f5b>